Hindawi
Security and Communication Networks
Volume 2022, Article ID 8783952, 13 pages
https://doi.org/10.1155/2022/8783952

Research Article

Audit as You Go: A Smart Contract-Based Outsourced Data Integrity Auditing Scheme for Multiauditor Scenarios with One Person, One Vote

Tengfei Li and Liang Hu

College of Computer Science and Technology, Jilin University, Changchun 130012, China

Correspondence should be addressed to Liang Hu; hul@jlu.edu.cn

Received 4 December 2021; Revised 14 January 2022; Accepted 14 February 2022; Published 28 March 2022

Academic Editor: Zhili Zhou

The data outsourcing services provided by cloud storage have greatly reduced the headache of data management for users, but the issue of remote data integrity poses further security concerns and computing burdens. The introduction of a third-party auditor (TPA) frees data owners from the auditing burden and alleviates disputes over the audit results between data owners and cloud storage providers. However, malicious cloud servers may collude with TPAs to deceive users for financial profits. Hiring multiple auditors in a single audit assignment appears to be a method to address the above problem, but the ensuing voting issues need to be further explored. In this paper, we proposed a smart contract-based outsourced data integrity auditing scheme for multiauditor scenarios. Unlike some existing schemes using reputation like factors as their voting weights, auditors in our scheme vote equally and audit as they go, without any maintenance. This mechanism not only frees auditors from trivia not related to the auditing but also avoids the drawbacks of centralization associated with over-high voting weights. The challenge used to check the integrity of the outsourced data is jointly generated by each involved auditor. Any collusion would be detected as long as there exists more than one honest auditor in the audit. We implement and deploy the scheme as Ethereum smart contracts. With the help of blockchain, the entire auditing process is public and transparent. Both the generated data and the obtained results are persisted with immutability, which ensures the traceability of all historical audits. The comprehensive theoretical and experimental analyses demonstrate that our scheme meets the claimed targets with high efficiency and low gas costs.

1. Introduction

With the rapid development of information age, individuals and organizations have produced a large amount of data. By 2025, the amount of data generated globally is expected to reach 463 exabytes each day [1]. Traditional local storage models can no longer meet the management needs of such a massive volume. Cloud storage is quickly attracting the attention of users for its scalability, low cost, and location free [2]. With technologies such as virtualization, cloud storage converges loose nodes into a powerful platform to provide unified services to users. Today, more and more people are willing to migrate their local data to leased cloud storage [3]. However, once the data is uploaded to the cloud storage, the owner completely loses control over the data. They are obliged to access the data through the interface provided by cloud storage servers, and they have to entirely rely on the cloud storage to ensure the integrity of their data. Unfortunately, even though cloud storage employs a variety of advanced technologies to guarantee the reliability and robustness of users' data, corruption caused by hardware failure, management errors, or external attacks still occurs [4]. What is worse, malicious servers may even delete the data that is rarely accessed by users in order to free up more storage space to gain greater profit. In addition, once data integrity has been compromised intentionally and otherwise, dishonest storage servers tend to conceal the incidents to prevent their reputation from tarnishing. So how to effectively detect the integrity of data stored in the cloud storage has become a research hotspot.

In order to address this problem, several remote data integrity auditing schemes have been proposed [5–12]. These schemes enable users to efficiently audit their data's integrity without a complete download. To achieve this, a user needs to divide the original file into blocks and then generate a tag for each block, which is used to verify the integrity of its corresponding block. When launching a file audit, a challenge will be generated and then sent to the storage server. The challenge contains a collection of selected block indexes and a collection of random numbers corresponding to the indexes. On receiving the challenge, the cloud server picks the data blocks specified in the challenge and computes them together with the random numbers to obtain an integrity proof. By verifying the proof, the data owner can determine whether the cloud server is actually keeping his data virgin or not.

To get rid of tedious audit routines and complex calculations, data owners would like to delegate TPAs to conduct audit tasks. However, introducing a third party poses additional risks that malicious cloud servers may try to trick their users by colluding with auditors. Employing multiple auditors on an audit assignment and determining the final audit outcome based on the votes of all participants can mitigate this collusion, but how to design a reasonable voting mechanism with multiple untrusted participants is still challenging.

The common method to deal with inconsistent voting results in a multiparticipant scenario is weighted voting, where a weight is supposed to be maintained for each auditor, which is typically represented by reputation. The weight of an auditor stands for the extent to which his vote influences the final result. In addition, when an auditor's vote is consistent with the final result, his reputation increases, otherwise it decreases. Intuitively, weighted voting hopes to build a virtuous ecosystem where honest auditors will always tell the truth and their reputation go on rising. On the contrary, dishonest auditors who are caught cheating will receive a reduction in reputation as punishment, and their reputation will keep declining as the cheating continues. At this rate, a few reputable auditors in the system are bound to become "elders," and their excessive voice will gradually centralize the system. In contrast to weighted voting, the result of nonweighted voting depends only on the number of votes each candidate receives, and the only thing that needs to be considered is membership, namely, who can vote and who is not allowed to vote in the system. If there is no threshold for the voting, malicious attackers can easily generate a large number of accounts with a very low cost to be involved in an audit, directly affecting the final result by an overwhelming numerical advantage. This type of attack is known as the Sybil attack [13], which is common on peer-to-peer networks.

In summary, the introduction of multiple auditors may somewhat mitigate the collusion, but the problem has not been fundamentally solved and the following threats remain.

(1) In weighted voting, the mechanism would lead the system to be progressively centralized. The collusion of a few reputable auditors is enough to sway the final outcome, even if honest auditors are outnumbered. This will reduce the cost of malicious cloud servers doing evil, while also weakening user confidence in the auditing system.

(2) In nonweighted voting, without a reasonable membership for the system, Sybil attacks can be easily launched. Malicious cloud servers can generate or buy large numbers of audit accounts to promote their desired results.

(3) The collusion between malicious cloud servers and auditors makes the detection of corrupted data fail. This collusion is undetectable because there is no way to distinguish whether an auditor's challenge is randomly generated or well constructed. With this, the cloud server can "truly" pass the proof verification by saving only a small part of specific data blocks.

1.1. Motivation.

1.1. Motivation. As mentioned above, in contrast to weighted voting, which inevitably leads to centralization, nonweighted voting only needs to design a reasonable membership mechanism to avoid Sybil attacks. Besides, the whole auditing process is considered to be written in the form of smart contracts and deployed to Ethereum, where any externally owned account (also known as a user account) can participate by simply paying a deposit. Another benefit of using smart contracts as the carrier for the multiauditor scenario is that it makes the process public, transparent, and traceable. The participants, details of the execution process, intermediate data, and the final result of the audit assignment are permanently recorded on the blockchain. You can always look up any historical audit without worrying about loss or manipulation.

1.2. Contribution. Based on the above motivation, we design a remote data integrity audit scheme based on Ethereum smart contracts with the following features:

(i) *Cheating resistance.* Without complete retention of user data, any server spoofing cannot pass the data integrity audit. This is the basic security requirement for remote data integrity auditing.

(ii) *Smart contract-based audit.* The auditing process is scheduled by smart contracts. Any Ethereum externally owned account can participate in the audit and has nothing to maintain, namely, audit as you go. Every single audit instance is persisted on the blockchain, which ensures public transparency and traceability.

(iii) *Collusion resistance.* We propose an aggregated challenge generation algorithm, where the final challenge is composed of the share independently submitted by each auditor. Such that, as long as there exists at least one honest auditor, the challenge is not going to be generated as malicious auditors might expect. We also designed a nonweighted voting mechanism, namely "one person, one vote." When the audit results come out inconsistent, the arbitration will be enforced and the honest will be rewarded and the dishonest punished.

1.3. Related Works. Traditional remote data integrity verification mechanisms fall into two main types: one is provable data possession (PDP) and the other is proof of retrievability (PoR). In PDP, the user requests a proof by sending some randomly selected blocks to the server and then determines the integrity of the remote data by verifying that the proof is correct. The PoR scheme stores each encrypted file in a cloud server with a set of pseudorandom blocks. The client can then check the integrity of the data by verifying that the server retains the pseudorandom blocks. In 2007, Ateniese et al. [5] first defined PDP and proposed the PDP scheme. In their scheme, the data user randomly selects several blocks of data to verify the integrity of the data with less communication and computational cost. If the integrity verification of these selected blocks passes, it can be determined that the server has a high probability of having complete data. Later, Juels et al. [14] proposed a PoR model in which the main idea is to embed a set of random values called "sentinels," and the auditor can check the integrity of the data by checking the presence of sentinels at specific data points. Shacham and Waters proposed two PoR schemes based on the homomorphic linear verifier [15], which further improved the efficiency of the PoR scheme proposed by Juels and Burton [14]. To implement PDP on dynamic cloud data, Ateniese et al. proposed another PDP scheme [16], which supports all dynamic operations except insertion operations. Shen et al. [6] proposed a dynamic PDP scheme that supports fully dynamic operations. Later, various PDP and PoR schemes were proposed to extend the performance or functionality of traditional schemes. A number of common PoR and PDP schemes have emerged to enrich the integrity checking capabilities of outsourced data, such as deduplication [17], batch audit [18, 19], and data update [7, 20]. To reduce the computational burden on the user side, public auditing schemes [8, 10–12, 21–23] are proposed to allow TPAs to audit the integrity of their cloud data on behalf of data owners. To guarantee the integrity of medical data and reduce the burden of the data owner, Li et al. [24] propose an efficient, privacy-preserving public auditing protocol for cloud-based medical storage systems that supports the functions of batch auditing and dynamic update of data. This scheme not only saves TPA and data owner computation costs but also reduces the communication overhead between TPA and cloud servers. Considering that key retention is a burden for data users, Shen et al. [25] propose a new paradigm called "data integrity auditing without private key storage," which utilizes a linear sketch with coding and error correction processes to confirm the identity of the user. To enable data integrity auditing under the multiwriter model, He et al. [26] propose the first public auditing scheme for shared data that supports fully dynamic operations. To implement the new paradigm, they proposed a specially designed authenticated structure, called the blockless Merkle tree, and a novel cryptographic primitive, called permission-based signature in edge computing scenarios, caching data on edge servers can minimize users' data retrieval latency. However, this new architecture poses challenges for traditional data audit models. Li et al. [27] propose a new data structure named variable Merkle hash tree (VMHT) for generating the integrity proofs of those data replicas during the audit, which solves the above problem. Considering existing schemes suffer from issues of complex certificate management or key escrow problems, Gudeme et al. [28] propose a certificateless privacy-preserving public auditing scheme for dynamic shared data with group user revocation in cloud storage, without public key infrastructure (PKI) or identity-based cryptography (IBC). To verify whether an untrusted CSP stores all their replicas in different geographic locations or not. Yu et al. [29] propose a dynamic multi-replica auditing scheme, with both the integrity and geographic locations of a cloud user's data replicas verified.

Recently, blockchain has been considered as one of the most promising technologies to provide security support for IoT systems [30]. It was initially used to provide digital payments [31] and is now commonly used for smart contracts [32, 33] and data storage. The trust issues associated with traditional data integrity verification make the integration of blockchain into data integrity verification an inevitable trend. Based on a distributed data storage blockchain, Zhang et al. [34] proposed a privacy-preserving electronic health record (EHR) public auditing scheme to prevent malicious behavior by TPA. However, it does not support batch auditing and data updates. Liu et al. [35] proposed to apply blockchain to avoid the use of TPA, and Yue et al. [36] proposed a blockchain-based framework that attempts to obtain trustworthy audit results. They all lack the necessary considerations to ensure the credibility of the results of off-chain events. Kun et al. [37] implemented private blockchain-based data validation in an untrustworthy environment, but their solution requires building and deploying a private blockchain, which is very difficult in practice. Zhou et al. [38] proposed a witnessing model to credibly enforce smart contract-based off-chain cloud service level agreements (SLA). Miao et al. [39] proposed a mechanism to generate challenges using block hashes, but the method does not guarantee that the audit results will not be tampered with off-chain. There are also some blockchain-based multiaudit models [37, 40]. However, their proof validation process is in smart contracts or in blockchains using proof of work, which can consume excessive costs of public chains or validation time. Zhang et al. [41] propose a certificateless public verification scheme against procrastinating auditors (CPVPA) by using blockchain technology. CPVPA is built on certificateless cryptography and is free from the certificate management problem. This scheme mitigates the impact of the TPA's laziness on the audit. To solve the problem of repeated auditing of data shared by multiple tenants, Xu et al. [42] propose a blockchain-based deduplicatable data auditing mechanism, which also works out the problems such as high cost and reliance on trusted third parties in traditional approaches. Chen et al. [33] proposed a blockchain-based crowdsourcing auditing approach to achieve trustworthiness in audit results. The model relies on an untrusted audit committee. However, the scheme maintains a reputation as the voting weight for each auditor, which may introduce the disadvantage of centralization to integrity auditing.

1.4. Organization. The rest of the paper is organized as follows. We discuss the preliminaries in Section 2. Section 3 describes the subalgorithms executed by each participant and the scheduling framework of the scheme. The security analysis and formal proof are described in Section 4. Section 5 analyses the implementation and performance. Finally, Section 6 concludes the paper.

2. Preliminaries

2.1. Bilinear Map. Let G and G_T be two multiplicative cyclic groups with a large prime order q. e: $G \times G \longrightarrow G_T$ is a bilinear map with the following properties:

(i) *Bilinearity*. $\forall u, v \in G$ and $\forall a, b \in Z_q^*$, it has $e(u^a, v^b) = e(u, v)^{ab}$.

(ii) *Non-degeneracy*. $\exists u, v \in G$ where u, v are generators of G, it has $e(u, v) \neq 1_G$.

(iii) *Computability*. $\forall u, v \in G$, there exists an efficient algorithm to calculate $e(u, v)$.

2.2. Complexity Assumption

Definition 1. (Computational Diffie–Hellman (CDH) problem). Suppose G is a multiplicative cyclic group. g is a generator of G. Given the tuple (g, g^a, g^b) with the unknown elements $a, b \in Z_q^*$, the CDH problem is to calculate g^{ab}.

Definition 2. (CDH assumption). The advantage for any probabilistic polynomial time (PPT) algorithm $\mathscr{A}$ to solve the CDH problem in G_1 is negligible. It is defined as $\mathrm{Adv}_{G_1\mathscr{A}}^{\mathrm{CDH}} = \Pr[\mathscr{A}(g, g^a, g^b) \longrightarrow g^{ab}\colon a, b \in_R Z_q^*] \leq \varepsilon$. Here, ε denotes a negligible value.

Definition 3. (Discrete logarithm (DL) problem). Given the tuple (g, g^a) where $a \in Z_q^*$ is unknown. the DL problem is to calculate a.

Definition 4. (DL assumption). The advantage for any PPT algorithm $\mathscr{A}$ to solve the DL problem in G_1 is negligible. It is defined as $\mathrm{Adv}_{G_1\mathscr{A}}^{\mathrm{DL}} = \Pr[\mathscr{A}(g, g^a) \longrightarrow a\colon a \in_R Z_q^*] \leq \varepsilon$. Here, ε denotes a negligible value.

2.3. Blockchain and Smart Contract. Blockchain technology enables decentralized peer-to-peer transactions, coordination, and collaboration without trust through data encryption, timestamps, and distributed consensus. A "smart contract" is simply a program that runs on the blockchain. It is a collection of codes (its functions) and data (its state) that resides at a specific address on the blockchain. They are typically used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without an intermediary's involvement or time loss. User accounts can interact with a smart contract by submitting transactions that execute a function defined in the smart contract. Smart contracts cannot be deleted by default, and interactions with them are irreversible. With the help of blockchain's immutability, the process of running smart contracts and generating data cannot be changed later. This is very important when you want to trust something or make something more trustable. The scheduling part of the audit assignments can be stripped out of the overall audit logic and put into a smart contract. The parties participate in the audit by interacting with the contract. The contract is responsible for driving the audit process, collecting the intermediate results of each participant's calculations, assigning calculation tasks to each participant, completing the vote tally, and outputting the final audit results.

2.4. Two-Phase Commit. The concept of two-phase commit (2PC) is derived from the database management system. It is a standardized protocol that ensures that a database commit is implemented in the situation where a commit operation must be broken into two separate parts. Since our audit scheme is based on smart contracts, any data submitted by the participants is publicly available. This poses a security risk to the operation of the protocol. The purpose of introducing 2PC in a public system is to ensure that the data submitted by each participant is confidential to others.

3. Proposed Scheme

In this section, we introduce the components of the proposed system and then explain the subalgorithms related to data integrity auditing and their executors.

3.1. System Model. The system consists of a data owner, a storage provider, an auditor, and a smart contract, where there can be any number of auditors.

(i) *Data Owner*. The data owner rent cloud storage services and outsource large amounts of data to the cloud storage. The data owner may be individual or organizational consumers.

(ii) *Storage Provider*. The storage provider provides cloud storage services to the data owner. It has significant storage capacity and powerful computing capability. When receiving a data auditing challenge, the storage provider should respond with an integrity proof to auditors.

(iii) *Auditor*. The auditor challenges the storage provider and identifies the integrity of the user data by verifying the proof returned by the provider.

(iv) *Smart Contract*. The smart contract stipulates the audit process. There are two smart contracts in the system. While AMSC (assignment management smart contract) manages the audit assignments, AASC (audit assignment smart contract) is instantiated by AMSC and performs a specific auditing assignment.

3.2. Notations. To make the proposed scheme more clearly understood, we summarize the main notations involved in Table 1.

Table 1: Notations.

Notation	Meaning
λ	A security parameter
G and G_T	Two cyclic multiplicative groups
g	The generator of G
e	A bilinear map
H and h	Two different hash functions
ϕ	A pseudorandom function
π	A pseudorandom permutation
u	A random value
x	The secret key
v	The public key calculated as g^x
F	A specific file to be outsourced
$F_{i\ d}$	An identity assigned to the above file F
m_i	One of the blocks constituting the file F
σ_i	One of the authenticators corresponding to m_i
Φ	The set of σ_i
t	The file tag of F
c	The number of challenged blocks
r and s	Two random numbers generated by each auditor, respectively
r_s and s_s	Two numbers aggregated by r and s, respectively, constituting the integrity challenge
μ and σ	Two data calculated by the challenge and the stored file, constituting the integrity proof

3.3. Auditing Framework. This section introduces how the smart contract boosts the interaction of each participant and achieves data security checks and aggregation. Note that all of them are executed on-chain except for the data outsourcing indicated by the dotted line in Figure 1, which is done off-chain.

(i) *Deploy AMSC*: AMSC is deployed at the very beginning. All storage providers, data owners, and auditors in the system are going to listen at its address for the events.

(ii) *Enroll File*: assuming data outsourcing has been done off-chain, the data owner submits the file identifier $F_{i\ d}$ and his storage provider's address to AMSC for the file enrollment.

(iii) *Request Audit*, *Instantiate a New AASC*, and *Inform Audit Information*: these three steps are done consecutively together. When an audit is launched, the data owner sends an auditing request to AMSC along with the fee he is willing to pay. The request includes $F_{i\ d}$ and the challenged number of data blocks c. After a brief verification, an AASC instance for this file will be deployed by AMSC. All participants listening to AMSC will receive this event. Consequently, the data owner and his storage provider begin to listen to the newly deployed AASC's address.

(iv) *Apply Audit*: at the same time, auditors also received the above event. Any interested auditor can apply for the audit by generating two large random numbers $r, s \in Z_p^*$ and submitting them to AASC with 2PC. Meanwhile, enough deposits are required. The detailed process of this 2PC is illustrated in Figure 2.

(v) *Challenge*: AASC adds all the submitted $r'\ s$ into r_s and $s'\ s$ into s_s, then sends $\{r_s, s_s\}$ to the corresponding storage provider as the challenge.

(vi) *Submit Proof*: on receiving the challenge, the storage provider computes $\{r_s, s_s\}$ together with the challenged data blocks to obtain the integrity proof $P = \{\mu, \sigma\}$.

(vii) *Verify Proof* and *Ballot*: after receiving P from the storage provider, AASC distributes it together with the two previously generated numbers $\{r_s, s_s, \mu, \sigma\}$ to auditors. Each auditor acquires the result of the data's integrity by checking the validity of P, then sends the result back to AASC.

(viii) *Judge*: AASC compares all the received results, if they are consistent, this result is taken as the final result. The balance in the contract account (including the data owner's auditing fee and the auditors' deposits deducted for failing 2PC) is then distributed to the remaining auditors as their rewards.

(ix) *Arbitrate*: if auditors do not draw a unanimous conclusion about the result, AASC sends $\{r_s, s_s, \mu, \sigma\}$ to the data owner, who then performs an arbitration to get the final auditing result. Based on this result, AASC distributes the balance in the contract account to the auditors who achieve the same result as the data owner as their rewards. The detailed process is illustrated in Figure 3.

3.4. Algorithms. This section introduces the calculations that each participant needs to complete in an auditing assignment.

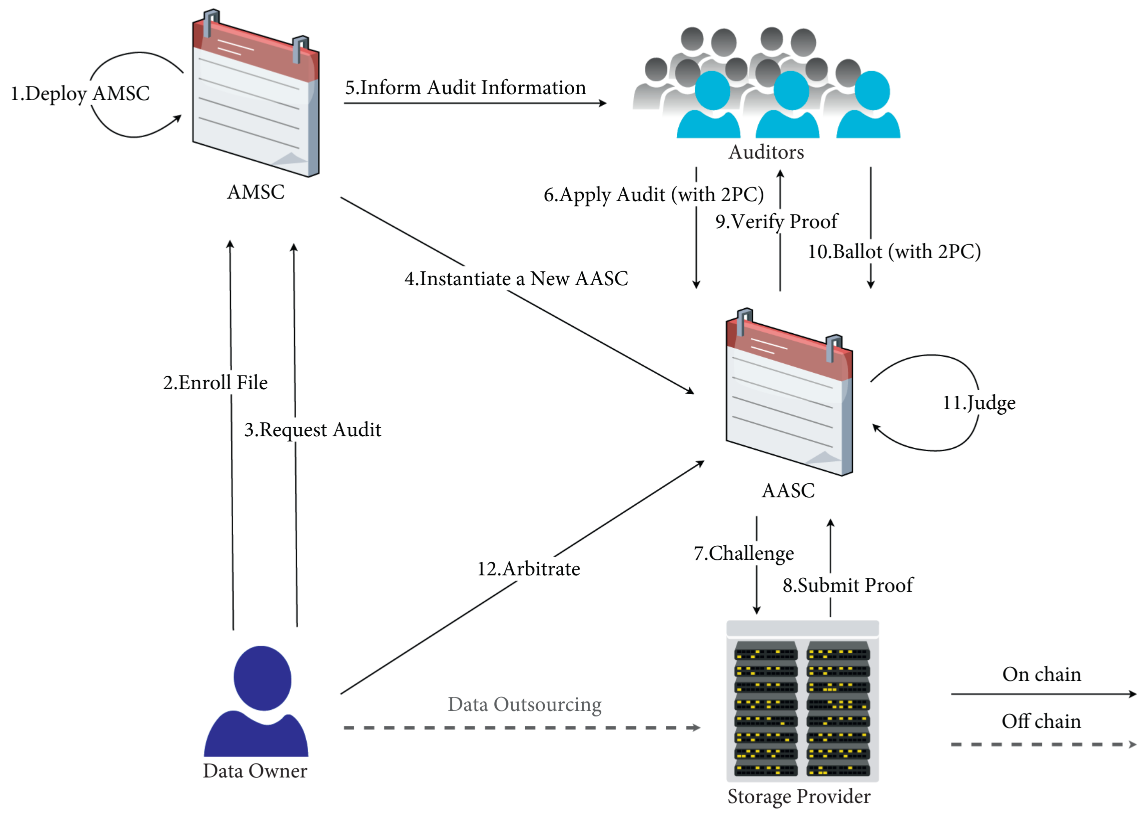

Figure 1: Smart contract-based audit process framework.

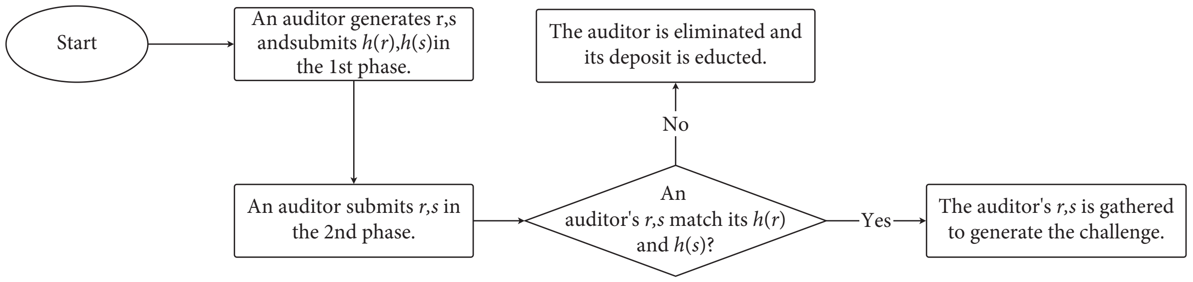
Start
An auditor generates r,s andsubmits $h(r)$,$h(s)$in the 1st phase.
The auditor is eliminated and its deposit is educted.
No
An auditor submits r,s in the 2nd phase.
An auditor's r,s match its $h(r)$ and $h(s)$?
Yes
The auditor's r,s is gathered to generate the challenge.

Figure 2: Two-phase commit process for applying an audit.

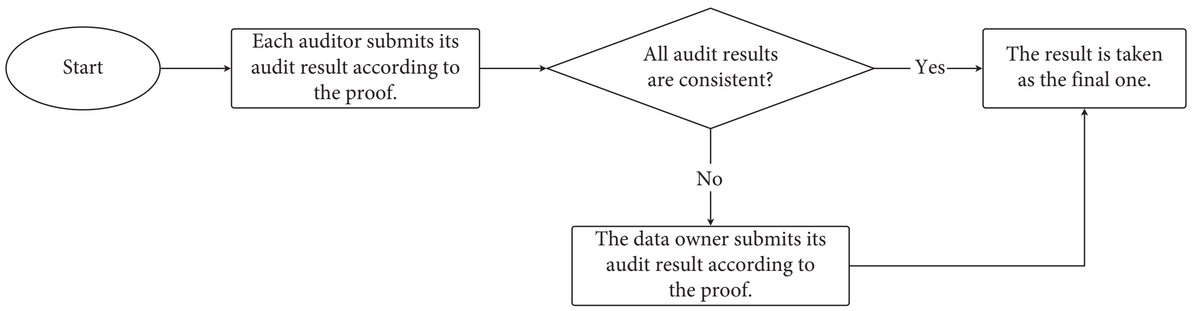
Start
Each auditor submits its audit result according to the proof.
All audit results are consistent?
Yes
The result is taken as the final one.
No
The data owner submits its audit result according to the proof.

Figure 3: Judgment and arbitration process for the final audit result.

(1) *Setup*$(1^\lambda) \longrightarrow \{pk, sk\}$: the algorithm is executed by the data owner. With the security parameter λ, the data owner chooses two cyclic multiplicative groups G, G_T with the same prime order $q > 2^k$, and one bilinear map e: $G \times G \longrightarrow G_T$. Let g be the generator of G. The data owner chooses a cryptographic hash function H: $\{0,1\}^* \longrightarrow G$, a pseudorandom function ϕ: $Z_q^* \times \{1, 2, \ldots, n\} \in Z_q^*$, a pseudorandom

permutation π: $Z_q^* \times \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, n\}$, a secure hash function h: $G_T \longrightarrow Z_p$, and a random value $u \in G$. Then the data owner selects a random value $sk = x \in Z_q^*$ as the secret key and calculate the public key as $v = g^x$. Finally, release $pk = \{q, G, G_T, e, H, h, \phi, \pi, u, v\}$ to public and keep sk as secret.

(2) $TagGen(F, F_{i\,d}, pk, sk) \longrightarrow \{\Phi, t\}$: this algorithm is executed by the data owner. Let the file $F = \{m_1, m_2, \ldots, m_n\}$ be identified by $F_{i\,d} \in Z_p^*$, where $m_i \in Z_q^*$ $(i = 1, \ldots, n)$. Then, for each block, the data owner computes the corresponding authenticator $\sigma_i = (H(W_i) \cdot u^{m_i})^x$, where $H(W_i)$ is the identifier of m_i and $W_i = F_{i\,d} \parallel i$. Then generate a file tag $t = F_{i\,d} \| \mathrm{Sig}_x(F_{i\,d})$, where $\mathrm{Sig}_x(F_{i\,d})$ is the signature of the file. The data owner then uploads the data file F and corresponding data tag $\{\Phi, t\}$ to the storage provider, where $\Phi = \{\sigma_i\}$.

(3) $TagVerify(F, \Phi, pk, F_{i\,d}) \longrightarrow \{0, 1\}$: this algorithm is executed by the storage provider. Besides verifying the validation of the file identifier's signature, the storage provider checks the correctness of each authenticator by

$$e(\sigma_i, g) = e(H(F_{i\,d} \| i) \cdot u^{m_i}, g^x), \quad (1)$$

and output the result of the authenticator verification, 1 for true and 0 for false.

(4) $Challenge(\cdot) \longrightarrow \{r_s, s_s\}$: this algorithm is executed by auditors together with AASC, each auditor independently picks two big random numbers r and s from Z_p^*, then sends them to AASC. AASC aggregates all r' s and s' s into r_s and s_s, respectively. AASC finally sends the two numbers to the storage provider as the data integrity challenge.

(5) $ProofGen(r_s, s_s, c, F, \Phi) \longrightarrow \{\mu, \sigma\}$: this algorithm is executed by the storage provider. On receiving r_s and s_s, together with the challenged block amount c, the storage provider calculates the challenge index set $I = \{u_i\}_{1 \le i \le c}$ and the random parameter set $\{v_i\}_{1 \le i \le c}$, where $u_i = \pi(r_s, i)$, $v_i = \phi(s_s, i)$. The storage provider sets $S = e(u, v)$, $\eta = h(S)$, $\mu' = \sum_{j \in I} m_j v_j$. After calculating $\mu = \eta\mu'$ and $\sigma = \prod_{j \in I} \sigma_j^{v_j}$, the storage provider responses $P = \{\mu, \sigma\}$ as the integrity proof to AASC.

(6) $ProofVerify(r_s, s_s, \mu, \sigma) \longrightarrow \{0, 1\}$: this algorithm is executed by each auditor. After receiving the proof $P = \{\mu, \sigma\}$, r_s and s_s transmitted by AASC, auditors check whether

$$e(\sigma^\eta, g) = e\left(\left(\prod_{j \in I} H(W_j)^{v_j}\right)^\eta \cdot u^\mu, g^x\right). \quad (2)$$

holds, then output the auditing result, 1 for true and 0 for false.

4. Analysis of Our Scheme

4.1. Security Model. We consider our scheme to fulfill the following two security requirements. First, the integrity of the challenged files is properly verified if the storage provider and auditors execute the protocol honestly. Second, the scheme resists semitrusted storage providers from deceiving the auditors about the integrity of the challenged data. It means, if the storage provider does not have the intact data file, it cannot generate the correct proof of data integrity. The first security requirement is defined as follows.

Definition 5. The proposed scheme is correct for data integrity checking, if for any random $r_s, s_s \in Z_p^*$, a data file F and the corresponding tag Φ, the following equation holds:

$$ProofVerify(r_s, s_s, ProofGen(r_s, s_s, F, \Phi)) = 1. \quad (3)$$

The second security requirement aims to resist three attacks mentioned in [43] launched by the storage provider, namely forge attack, replay attack, and replace attack. In each of these three attacks, the semitrusted storage provider responds to auditors with an invalid proof. We can capture the requirement through a security game that covers all three attacks. This security game consists of adversary $\mathcal{A}$ and challenger $\mathcal{C}$. $\mathcal{A}$ plays the role of a semitrusted storage provider who tries to trick auditors by forging data integrity proof. The game is described as follows:

(1) $\mathcal{C}$ runs Setup(1^λ) algorithm to generate $\{pk, sk\}$, then release pk to $\mathcal{A}$.

(2) $\mathcal{A}$ makes queries repeatedly to $\mathcal{C}$ for some files. $\mathcal{C}$ returns $\Phi \leftarrow \mathrm{TagGen}(F, F_{i\,d}, pk, sk)$ to $\mathcal{A}$.

(3) Finally, $\mathcal{A}$ outputs $\{\sigma, \mu\}$ for a data file F and data tag Φ on the challenge $\{r_s, s_s\}$.

We define the advantage of $\mathcal{A}$ is $\mathrm{Adv}_{\mathcal{A}} = \Pr[\mathrm{ProofVerify}(r_s, s_s, \mu, \sigma) = 1]$. We say the adversary wins the above game if $\mathrm{Adv}_{\mathcal{A}}$ is non-negligible.

Definition 6. The proposed scheme is sound, if there exists an efficient extraction algorithm such that, for $\{\sigma, \mu\}$ output by adversary $\mathcal{A}$ to the data file F and data tag Φ on the challenge $\{r_s, s_s\}$ and $\mathcal{A}$ wins the above game, the extraction algorithm recovers file F from Φ and $\{\sigma, \mu\}$.

4.2. Security Analysis

Theorem 1. *(Auditing correctness). When the storage provider stores the user's data correctly, the proof it generates can be verified by auditors.*

Proof. Given valid proof from the storage provider $P = \{\mu, \sigma\}$, the verification equation (1) in the $ProofVerify$ algorithm will hold. Based on the properties of the bilinear mapping, the verification equation (1) can be proved correct by deriving the left-hand side from the right-hand side as follows:

$$\begin{aligned}
e(\sigma^{\eta}, g) &= e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta} \cdot u^{\mu}, g^{x}\right)\\
&= e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta} \cdot u^{\eta \sum_{j\in I} m_j \cdot v_j}, g^{x}\right)\\
&= e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta} \cdot \left(\prod_{j\in I} u^{m_j\cdot v_j}\right)^{\eta}, g^{x}\right)\\
&= e\left(\left(\prod_{j\in I} \left(H(W_j)\cdot u^{m_j}\right)^{v_j}\right)^{\eta}, g^{x}\right)\\
&= e\left(\left(\prod_{j\in I} \left(H(W_j)\cdot u^{m_j}\right)^{x\cdot v_j}\right)^{\eta}, g\right)\\
&= e\left(\left(\prod_{j\in I} {\sigma_j}^{v_j}\right)^{\eta}, g\right)\\
&= e(\sigma^{\eta}, g).
\end{aligned} \tag{4}$$

We use the hybrid argument technique to prove soundness, as in [15]. "Hybrid arguments" have been used extensively in cryptography for many years. Such an argument is essentially a sequence of transitions based on indistinguishability. First of all, we define the following games: Game-0. Game-0 is the original game defined in Section 4.1.

Game-1. Game-1 is the same as Game-0, except that the challenger $\mathscr{C}$ keeps a local list of all the tags he has signed. If the adversary $\mathscr{A}$ has ever submitted a tag Φ that (2) has a valid signature under *sk* but (2) has not been signed by $\mathscr{C}$, then $\mathscr{C}$ announces failure and aborts.

Game-2. Game-2 is the same as Game-1, except that $\mathscr{C}$ records all responses to TagGen queries from $\mathscr{A}$. If $\mathscr{A}$ succeeds but σ output by $\mathscr{A}$ is not equal to $\prod_{j\in I} {\sigma_j}^{v_i}$, the challenger $\mathscr{C}$ announces failure and aborts.

Game-3. Game-3 is the same as Game-2, except that challenger $\mathscr{C}$ announces failure and aborts if at least one $\mu' \neq \sum_{j\in I} v_j \cdot m_j$.

Lemma 1. *If there exists an algorithm $\mathscr{A}$ that can distinguish between Game-0 and Game-1 with a non-negligible probability, then we can construct an algorithm $\mathscr{B}$ to break the existential unforgeability with non-negligible advantage.*

Analysis. If $\mathscr{A}$ causes $\mathscr{C}$ to abort in Game-1, then we can use $\mathscr{A}$ to construct an algorithm $\mathscr{B}$ against the existential unforgeability of the signature scheme.

Lemma 2. *If there exists an algorithm $\mathscr{A}$ that can distinguish between Game-1 and Game-2 with a non-negligible probability, then we can construct an algorithm $\mathscr{B}$ to break the computational Diffie–Hellman assumption with non-negligible advantage.*

Analysis. Suppose that g^x and g^y are elements of the CDH problem and we set $v = g^x, u = g^y$. Suppose $\mathscr{A}$ can respond to a signature σ', which is different from the expected signature σ. We can compute

$$e\left(\frac{\sigma'}{\sigma, g}\right) = e\left(\prod_{j\in I} u^{\Delta\mu_j}, v\right) = e\left(g^{\sum_{j\in I}\Delta\mu_j\cdot x\cdot y}, g\right). \tag{5}$$

Therefore, we can calculate $g^{x\cdot y} = (\sigma'/\sigma)^{1/\sum_{j\in I}\Delta\mu_j}$.

Lemma 3. *If there exists an algorithm $\mathscr{A}$ that can distinguish between Game-2 and Game-3 with a non-negligible probability, then we can construct an algorithm $\mathscr{B}$ to break the computational Diffie–Hellman assumption with non-negligible advantage.*

Analysis. We assume that $h(\cdot)$ is a random oracle controlled by an extractor that answers a hash query posed by the adversary. For $\eta = h(S)$ from the extractor, the adversary outputs $\{\mu, \sigma\}$ such that

$$e(\sigma^{\eta}, g) = e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta} \cdot u^{\mu}, v\right). \tag{6}$$

Then, the extractor sets $h(S)$ to be $\eta^* \neq \eta$. The adversary outputs $\{\mu^*, \sigma\}$ such that

$$e\left(\sigma^{\eta^*}, g\right) = e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta^*} \cdot u^{\mu^*}, v\right). \tag{7}$$

We divide the above two equations, then we have

$$\begin{aligned}
e\left(\sigma^{\eta-\eta^*}, g\right) &= e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta-\eta^*} \cdot u^{\mu-\mu^*}, v\right)\\
e\left(\sigma^{\eta-\eta^*}, g\right) &= e\left(\left(\prod_{j\in I} H(W_j)^{v_j}\right)^{\eta-\eta^*} \cdot u^{\mu-\mu^*}, g^{x}\right)\\
\sigma^{\eta-\eta^*} &= \left(\prod_{j\in I} H(W_j)^{v_j}\right)^{x(\eta-\eta^*)} \cdot u^{x(\mu-\mu^*)}\\
\left(\prod_{j\in I} {\sigma_j}^{v_j}\right)^{\eta-\eta^*} &= \left(\prod_{j\in I} H(W_j)^{v_j}\right)^{x(\eta-\eta^*)} \cdot u^{x(\mu-\mu^*)}\\
u^{x(\mu-\mu^*)} &= \left(\prod_{j\in I} \left(\sigma_j/H(W_j)^{x}\right)^{v_j}\right)^{\eta-\eta^*}\\
u^{x(\mu-\mu^*)} &= \left(\prod_{j\in I} \left(u^{x m_j}\right)^{v_j}\right)^{\eta-\eta^*}\\
\mu-\mu^* &= \left(\sum_{j\in I} m_j v_j\right)\cdot(\eta-\eta^*)\\
\sum_{j\in I} m_j v_j &= \frac{(\mu-\mu^*)}{(\eta-\eta^*)}.
\end{aligned} \tag{8}$$

Finally, $\{\sigma, \mu' = (\mu - \mu^*)/(\eta - \eta^*)\}$ can be taken as a response to the extractor.

Theorem 2. *(Soundness). Assume that the computational Diffie–Hellman problem is hard in bilinear groups and the digital signature scheme is existentially unforgeable. Then no probabilistic polynomial-time adversary can break the soundness of the scheme with a non-negligible probability.*

Proof. Any adversary's advantage in Game 3 must be 0, because if there is no intact file F, i.e., at least one $\mu' \neq \sum_{j\in I} m_j v_j$, the challenger always announces failure and aborts. According to the game sequence and Lemmas 1–3, the advantage of the adversary in the original game, Game 0 must be negligible. □

4.3. Analysis of Collusion Resistance. As portrayed in the $ProofGen$ algorithm, r_s is used as the seed for the pseudorandom function ϕ to generate the indexes of the blocks to be challenged. This means that if ϕ is inherently secure, the indexes cannot be known without knowing r_s. As proven in Theorem 2, the probability that a storage provider generates a proof that passes the verification without preserving the complete data is negligible. Malicious auditors can make the negotiated seed fall into the designed set by colluding with the storage provider, so that the indexes and random numbers of the challenge blocks are generated as per their expectation. The storage provider only needs to store a small part of the real data block to pass the ProofVefiry algorithm. As long as there exists at least one honest auditor involved in the audit, the generation of aggregated random numbers is then not controlled by malicious auditors, and the probability that the number happens to be in the designed set is $w/|Z_p^*|$, where w is the size of the set, which is negligible.

4.4. Discussion on Data Owner's Trustworthy. The only security assumption in our scheme is that the data owner is honest. The data owner will perform arbitration when the auditors do not reach a consensus on the audit results. Actually, this is different from cutting out the auditors and allowing the data owner to perform the audit directly by himself. In a system where only two parties participate, the conclusions declared by either party are unconvincing. In our scheme, the arbitration will only be performed when the auditors' results are inconsistent, which means that each kind of result is reached by multiple individuals. Moreover, the auditor who lied will definitely be discovered, which allows the data owner to perform the audit directly by himself. This leaves the auditor with no reason to lie, meaning that the arbitration may rarely be enforced.

4.5. Discussion on the Employability of Two-phase Commit. Our program uses 2PC in two phases, *Challenge Generation* and *Result Submission.* The generation of the challenge in our scheme relies on two numbers submitted by each auditor independently, which are confidential to the other auditors. If a malicious auditor knows other auditors' numbers, then he can construct special numbers that prompt the smart contract to generate a challenge as he intends, which will make the whole scheme fail. We artificially divide the submission of secret numbers into two steps by introducing 2PC: the first step submits the hash value, and the second step submits the corresponding hash key. Due to the one-way nature of hashing, a malicious auditor cannot derive the secret numbers in the first step even if he knows the hash value, and thus cannot have any influence on the generation of the final challenge by constructing his own number. When it comes to the results submission phase, some auditors may choose to copy other auditors' results due to their laziness. In the first step, the honest auditor can concatenate the audit result with its blockchain address to calculate the hash value. In this way, the smart contract can determine whether an auditor has copied someone else's results by checking whether the hash value submitted in the second step matches the key submitted in the first commit.

5. Implementation and Performance Analysis

In this section, we discuss the performance of the proposed scheme in terms of computation and gas cost, respectively. We carry out a series of simulation experiments to evaluate the performance of our scheme, and the codes can be found at https://github.com/TDMaker/sc-paper. Note that, since the underlying layer of our scheme is a P2P overlay network, the network traffic required to maintain it must be much larger than other end-to-end schemes, so we have omitted the comparison of communication costs.

5.1. Environment. The experiments were carried out on an Ubuntu Desktop 20.04 with the processor of Intel(R) Core(TM) i7-6500U CPU @ 2.50 GHz × 4 and 4 GB of RAM. In the local computing part of each participant, we use the pairing-based cryptography (`PBC`) library [44] and the GNU multiple precision arithmetic (`GMP`) [45], and we implement the simulation experiment using `C` language. In our experiments, we choose the parameter `a.param` to be the parameters of the `PBC` library. The smart contracts are written in *Solidity* language and run in the *Rinkeby* Ethereum test net. Each participant uses programs written in `JavaScript` to interact with the smart contract by calling the `npm`: `web3` package [46].

5.2. Computation Analysis. We analyze the computation costs of all subalgorithms of the proposed protocol. We chose the size of the data block to be 160 bits. Without loss of generality, we change the block count from 100 to 1000 with an increment of 100 in each test. Since $TagGen$ and $TagVerify$ are executed only once for the same file, and the time overhead is relatively large compared with other algorithms, as shown in Figure 4. The rest of the algorithms need to be executed repeatedly during each audit, as shown in Figure 5. For $Setup$, it is used to generate the system parameters. Since its time overhead is static and relatively small, we do not plot it on the figure and only note it 4.715 ms averaged over ten experiments.

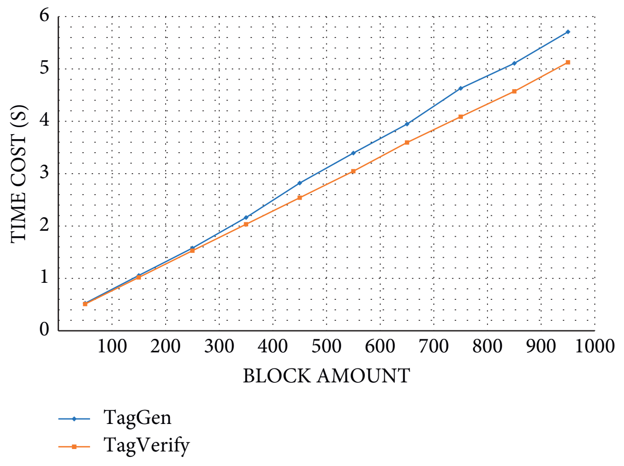

Figure 4: Time overhead of *TagGen* and *TagVerify* as the amount of blocks increases.

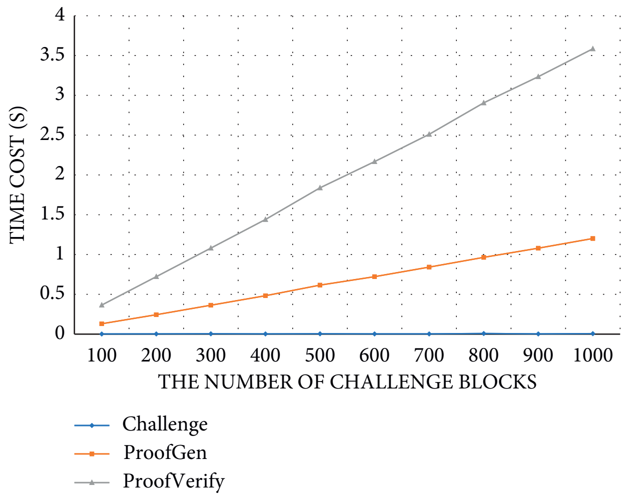

Figure 5: Time overhead of *Challenge*, *ProofGen*, and *ProofVerify* as the number of challenged blocks increases.

For *TagGen* and *TagVerify*, it is used to compute and check data owner's outsourced authenticators, which take much longer time than other algorithms. This time cost increases with the size of the user file, which might become quite large. Fortunately, this time cost is one-time and can be done offline. For *Challenge*, it is to determine two random numbers to constitute the challenged block's index sequence, which is pretty fast. For *ProofGen*, it is to calculate integrity proofs by aggregating the challenged data blocks. This time cost relies mainly on the length of the challenge sequence and increases with the number of challenged blocks. For *ProofVerify*, it is to check the integrity proof, which is generated by the storage provider. This time cost is also increasing with the number of challenged blocks due to the same reason as *ProofGen*. In our protocol, the most frequent algorithms are *Challenge*, *ProofGen*, and *ProofVerify*, which are periodically performed by the storage provider and auditors. Thus, data owners in our protocol have a little workload after data outsourcing except when arbitration is needed.

Comparison: to show the efficiency advantage of our scheme, we compare it with the schemes proposed in [24, 25, 47–49]. We list the results in Table 2. The computation cost of our scheme mainly lies in the expensive operations such as multiplication, exponentiation, and pairing. Other operations like hash function and addition only incur negligible costs, so we omit them when analyzing the computation cost. For simplicity, we use T_{mul}, $T_{\exp}$, and T_p to represent the overhead of multiplication operation, exponentiation operation, and pairing operation on group G, respectively. Suppose there are n blocks in total, of which c blocks are challenged. It is easy to see that the entire efficiency of the scheme is mainly dependent on the efficiency of the algorithms *TagGen*, *TagVerify*, *ProofGen*, and *ProofVerify*. However, the *TagGen* and *TagVerify* are run only once, its impact on the overall efficiency of the audit protocol is negligible. Therefore, we only make comparisons to evaluate the efficiency of the algorithms *ProofGen* and *ProofVerify*. It is easy to find that our scheme takes the same number of multiplication operations as [48], but one more than all

Table 2: Computation comparison with some existing schemes.

Schemes	Proof generation	Proof verification
Scheme in [47]	$T_p + (c+1)T_{\exp} + (c-1)T_{mul}$	$2T_p + (c+1)T_{\exp} + (c+1)T_{mul}$
Scheme in [48]	$2T_p + (c+1)T_{\exp} + cT_{mul}$	$cT_p + (c+1)T_{\exp}$
Scheme in [49]	$(c+1)T_{\exp} + (c-1)T_{mul}$	$3T_p + (c+3)T_{\exp} + (c+1)T_{mul}$
Scheme in [25]	$cT_{\exp} + (c-1)T_{mul}$	$2T_p + (c+2)T_{\exp} + cT_{mul}$
Scheme in [24]	$2T_p + (c+2)T_{\exp} + (c-1)T_{mul}$	$T_p + 3T_{\exp} + cT_{mul}$
Our scheme	$(c+1)T_{\exp} + cT_{mul}$	$2T_p + (c+2)T_{\exp}$

four remaining schemes in proof generation. Our scheme has the same exponentiation operation as the first three schemes. Meanwhile, [25] has one less exponentiation operation than ours, and [24] has one more than ours. The pairing operation is the most time-consuming operation, but it occurs only in [24, 47, 48]. In proof verification, the scheme [25, 47] needs two pairing operations and the scheme [49] needs three paring operations, but in the scheme [48], the paring operation is linear with the number of challenged blocks, while our scheme and the scheme [48] reduces $(c+1)$ multiplication operation and two exponentiation operations compared with the scheme [47, 49]. Although [24] outperforms other schemes in terms of exponentiation and paring operations, it does increase linearly in terms of multiplication operations.

Nonetheless, the above schemes all make various computational concessions for functionality while satisfying their proposed functional properties on the basis of security. So, the mere computation cost comparison can only be used as a meager reference.

5.3. Gas Cost Analysis. Gas is the fuel to be paid for running smart contracts on Ethereum. It measures how much "work" needs to be done for an operation or a series of operations. The gas prevents junk transactions from blocking the network and serves as additional income for miners. We deployed our smart contracts on Rinkeby [50], which is an Ethereum test net (or test network). The only difference in whether all auditors reach the same conclusion is that there is an additional step of *arbitrate* by the data owner at the end. All other steps are exactly the same. Therefore, we only explain the case that requires the data owner's arbitration. Because any number of lying auditors can be detected as long as there exists at least one honest auditor, and because the number of dishonest auditors have no effect on the final result, we introduce only two auditors in the experiment: an honest auditor and a dishonest one.

Figure 6 illustrates such an audit assignment, where the vertical coordinates represent each participant, and the horizontal coordinates portray how much Wei of gas each participant spends to execute a certain algorithm. Wei is the smallest unit of currency in Ethereum. 1 Ether = 10^{18} Wei. Note that, $Inform Submit Hash Key$ 1, $Inform Submit Hash Key$ 2, and $Inform Proof Gen$ are events emitted (which are only auxiliary steps), so we did not list them in Section 3.3 for clarity. *Submit Hash Key* 1 and *Submit Hash Key* 2 are substeps of 2PC, so they have

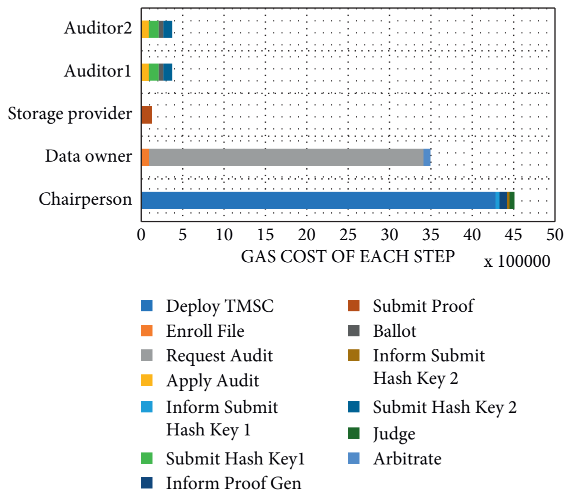

Figure 6: Gas cost of each step.

not been listed, either. As we can see from the figure, the two algorithms with the highest gas cost are *Deploy TMSC* and *Request Audit*, because these two algorithms involve the deployment of smart contracts. The large amount of gas consumed by smart contract deployment comes from two aspects, on the one hand, the CREATE op code of the smart contract, which is called during contract creation, costs a fixed 32,000 gas; on the other hand, from the storage of contracts, more byte code means more storage, and each byte costs 200 gas. This adds up very quickly. And the left operations require very little gas overhead. Fortunately, the *Deploy TMSC* algorithm is a management contract for audit assignments that is deployed only once in an audit system, while the *Request Audit* algorithm is instantiated once for every audit assignment executed. Other operations require less gas overhead. In an audit assignment, the increase of gas for each additional auditor is less than 400,000. The gas overhead for the other participants is fixed, except when data owner arbitration is required, which needs an additional 100,000 gas, but this overhead is insignificant compared to the reward it can earn.

6. Conclusion

In this paper, we design a remote data integrity audit auditing scheme based on the Ethereum smart contract. The challenge of this scheme is jointly generated by all Ethereum users participating in the audit. When auditing results are inconsistent, the data owner will complete the final arbitration. Safety proofs and experimental results show that our scheme is secure and efficient.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest with this study.

Acknowledgments

This work is funded by: the National Natural Science Foundation of China under grant no. 61701190, by the National Key R&D Plan of China under grant no. 2017YFA0604500, and by the National Sci-Tech Support Plan of China under grant no. 2014BAH02F00.

References

[1] B. Vuleta, "How Much Data Is Created Every Day?" 2021.

[2] I. Upadhyay, "Top 10 Major Characteristics of Cloud Computing," 2020.

[3] F Joseph, "Kovar. 8 Emerging Data Storage Trends to Watch in," 2021.

[4] R. Paul, "8 Cloud Storage Problems and How to Avoid Them," 2018.

[5] Giuseppe Ateniese, R. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 598–609, Alexandria, VA, USA, Octobar, 2007.

[6] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2402–2415, 2017.

[7] Z. Ren, L. Wang, Q. Wang, and M. Xu, "Dynamic proofs of retrievability for coded cloud storage systems," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 685–698, 2015.

[8] L. Chang, R. Ranjan, X. Zhang, C. Yang, D. Georgakopoulos, and J. Chen, "Public auditing for big data storage in cloud computing–a survey," in *Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering*, pp. 1128–1135, IEEE, Sydney, Australia, December, 2013.

[9] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proceedings of the 2010 proceedings ieee infocom*, vol. 1–9, March, 2010.

[10] G. Yang, J. Yu, W. Shen, Q. Su, Z. Fu, and R. Hao, "Enabling public auditing for shared data in cloud storage supporting identity privacy and traceability," *Journal of Systems and Software*, vol. 113, pp. 130–139, 2016.

[11] Yu. Jia, K. Ren, and C. Wang, "Enabling cloud storage auditing with verifiable outsourcing of key updates," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1362–1375, 2016.

[12] J. Yuan and S. Yu, "Efficient public integrity checking for cloud data sharing with multi-user modification," in *Proceedings of the IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 2121–2129, IEEE, Toronto, ON, Canada, May, 2014.

[13] J. R. Douceur, "The sybil attack, Peer-to-Peer Systems," in *Proceedings of the International Workshop on Peer-To-Peer Systems*, pp. 251–260, Springer, Cambridge, MA, USA, March, 2002.

[14] A. Juels and S. K. Burton, "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 584–597, Alexandria, VA, USA, Octobar, 2007.

[15] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the International conference on the theory and application of cryptology and information security*, pp. 90–107, Springer, Melbourne, Australia, December, 2008.

[16] Giuseppe Ateniese, R. Di Pietro, L. V Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pp. 1–10, Istanbul, Turkey, September, 2008.

[17] J. Wu, Y. Li, T. Wang, and Y. Ding, "Cpda: a confidentiality-preserving deduplication cloud storage with public cloud auditing," *IEEE Access*, vol. 7, Article ID 160482, 2019.

[18] Y. Fan, X. Lin, G. Tan, Y. Zhang, W. Dong, and J. Lei, "One secure data integrity verification scheme for cloud storage," *Future Generation Computer Systems*, vol. 96, pp. 376–385, 2019.

[19] J. Han, Y. Li, and W. Chen, "A lightweight and privacy-preserving public cloud auditing scheme without bilinear pairings in smart cities," *Computer Standards & Interfaces*, vol. 62, pp. 84–97, 2019.

[20] H. Jin, H. Jiang, and Ke Zhou, "Dynamic and public auditing with fair arbitration for cloud data," *IEEE Transactions on cloud computing*, vol. 6, no. 3, pp. 680–693, 2016.

[21] W. Shen, J. Yu, H. Xia, H. Zhang, X. Lu, and R. Hao, "Lightweight and privacy-preserving secure cloud auditing scheme for group users via the third party medium," *Journal of Network and Computer Applications*, vol. 82, pp. 56–64, 2017.

[22] Jia Yu, K. Kui Ren, C. Cong Wang, and V. Varadharajan, "Enabling cloud storage auditing with key-exposure resistance," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 6, pp. 1167–1179, 2015.

[23] Yu. Jia and H. Wang, "Strong key-exposure resilient auditing for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1931–1940, 2017.

[24] Li. Xiong, S. Liu, R. Lu, M. K. Khan, Ke. Gu, and X. Zhang, "An efficient privacy-preserving public auditing protocol for cloud-based medical storage system," *IEEE Journal of Biomedical and Health Informatics*, vol. 1, no. –1, 2022.

[25] W. Shen, J. Qin, Yu. Jia, H. Rong, J. Hu, and J. Ma, "Data integrity auditing without private key storage for secure cloud storage," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, p. 2021.

[26] K. He, J. Chen, Y. Quan, S. Ji, D. He, and R. Du, "Dynamic group-oriented provable data possession in the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1394–1408, 2021.

[27] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1210–1223, 2021.

[28] J. R. Gudeme, S. Pasupuleti, and R. Kandukuri, "Certificateless privacy preserving public auditing for dynamic shared data with group user revocation in cloud storage," *Journal of*

Parallel and Distributed Computing, vol. 156, pp. 163–175, 2021.

[29] H. Yu, Z. Yang, M. Waqas et al., “Efficient dynamic multi-replica auditing for the cloud with geographic location,” *Future Generation Computer Systems*, vol. 125, pp. 285–298, 2021.

[30] H. Chen, J. Yu, H. Zhou, T. Zhou, F. Liu, and Z. Cai, “Smartstore: a blockchain and clustering based intelligent edge storage system with fairness and resilience,” *International Journal of Intelligent Systems*, vol. 36, no. 9, pp. 5184–5209, 2021.

[31] M. Swan, *Blockchain: Blueprint for a New Economy*, O’Reilly Media, Inc., Sebastopol, California, 2015.

[32] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[33] H. Chen, H. Zhou, J. Yu et al., “Trusted audit with untrusted auditors: a decentralized data integrity crowdauditing approach based on blockchain,” *International Journal of Intelligent Systems*, vol. 36, 2021.

[34] X. Zhang, J. Zhao, C. Xu, H. Li, H. Wang, and Y. Zhang, “Cipppa: conditional identity privacy-preserving public auditing for cloud-based wbans against malicious auditors,” *IEEE Transactions on Cloud Computing*, vol. 9, 2019.

[35] B. Liu, S. Chen, X. Xu, and L. Zhu, “Blockchain based data integrity service framework for iot data,” in *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 468–475, IEEE, Honolulu, HI, USA, June, 2017.

[36] D. Yue, R. Li, Y. Zhang, W. Tian, and C. Peng, “Blockchain based data integrity verification in p2p cloud storage,” in *Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 561–568, IEEE, Singapore, December, 2018.

[37] H. Kun, J. Xin, Z. Wang, and G. Wang, “Outsourced data integrity verification based on blockchain in untrusted environment,” *World Wide Web*, vol. 23, no. 4, pp. 2215–2238, 2020.

[38] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao, “A blockchain based witness model for trustworthy cloud service level agreement enforcement,” in *Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1567–1575, IEEE, Paris, France, May, 2019.

[39] Y. Miao, Q. Huang, M. Xiao, and H. Li, “Decentralized and privacy-preserving public auditing for cloud storage based on blockchain,” *IEEE Access*, vol. 8, Article ID 139813, 2020.

[40] Y. Du, H. Duan, A. Zhou, C. Wang, M. Ho Au, and Q. Wang, “Towards privacy-assured and lightweight on-chain auditing of decentralized storage,” in *Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pp. 201–211, IEEE, Singapore, December, 2020.

[41] Y. Zhang, C. Xu, X. Lin, and X. Shen, “Blockchain-based public integrity verification for cloud storage against procrastinating auditors,” *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 923–937, 2021.

[42] Y. Xu, C. Zhang, G. Wang, Z. Qin, and Q. Zeng, “A blockchain-enabled deduplicatable data auditing mechanism for network storage services,” *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1421–1432, 2021.

[43] H. Yan, J. Li, J. Han, and Y. Zhang, “A novel efficient remote data possession checking protocol in cloud storage,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 78–88, 2016.

[44] B. Lynn, “Pbc Library - Pairing-Based Cryptography,” 2013.

[45] Gmp, “The Gnu Mp Bignum Library,” 2020.

[46] Web3, “web3.js - Ethereum Javascript Api,” 2021.

[47] C. Wang, S. SM. Chow, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for secure cloud storage,” *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2011.

[48] Y. Yu, M. H. Au, A. Giuseppe et al., “Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 767–778, 2016.

[49] J. Li, H. Yan, and Y. Zhang, “Identity-based privacy preserving remote data integrity checking for cloud storage,” *IEEE Systems Journal*, vol. 15, no. 1, pp. 577–585, 2020.

[50] R. Rinkeby, “Ethereum Testnet,”.